CSE 154: Web Programming — Autumn 2018

Final Exam

Name: _____

UWNet ID: _____@uw.edu

TA (or section): _____

Rules:

- You have 110 minutes to complete this exam.
- Do not open this booklet until time has begun. You will receive a deduction if you open the booklet before time is started, or if you keep working after the instructors call for papers.
- You may not use any electronic or computing devices, including calculators, cell phones, smartwatches, and music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- Do not abbreviate code, such as writing ditto marks (”) or dot-dot-dot marks (...).
- The only functions you may assume are defined are $ (for `document.getElementById`), qs (for `document.querySelector`), qsa (for `document.querySelectorAll`), and `checkStatus`.
- You must show your Student ID to a TA or instructor while sitting in your assigned seat in order to have received this exam.
- Once you enter the room, you must turn all parts of the exam with your name on it and will not be permitted to leave without doing so.
- There is an extra piece of paper stapled to the back of this exam. If you continue problem, please make sure to note which problem you are continuing back there
- It is your responsibility to ensure your exam is stapled back together and/or your name is on every page if you choose to pull the staple out of this booklet.

Question	**Score**	**Possible**
HTML/CSS		15
Short Answer		18
JS/DOM/Animations		12
JS with AJAX		20
PHP Web Service		20
SQL Queries		15
Extra Credit		1

This page intentionally left blank

1.A. Debug * the Things! (CSS Query Selectors, 5pts)

Consider the following HTML body:

```
<body>
  <header id="title-1">
    <h1 id="title-2">How to Debug <strong id="strong-1">'em</strong> All!</h1>
  </header>
  <main>
    <h2 id="title-2">Tips:</h2>
    <ul id="list-1">
      <li id="tip-1">
        <strong id="strong-2">Never</strong> underestimate the subtle semicolon...
      </li>
      <li id="tip-2">The JS console is your best friend!</li>
      <li id="tip-3">Talk through your code with a swaggy rubber duck</li>
      <li id="tip-4" class="fun">
        Sometimes, you just need to take a break from code...
        <ol id="list-2" class="fun">
          <li id="sub-1">Pet a puppy</li>
          <li id="sub-2">Solve a puzzle</li>
          <li id="sub-3">Take a nap <strong id="strong-3">with a puppy</strong></li>
        </ol>
      </li>
    </ul>
    <p id="p-1">
      <img id="img-1" src="caterpie.jpg" alt="lucky debugging Caterpie"/>
    </p>
  </main>
</body>
```

Write the ID's of the elements selected by each of the given selectors:

1. `ul strong`

_____

2. `.fun`

_____

3. `li li`

_____

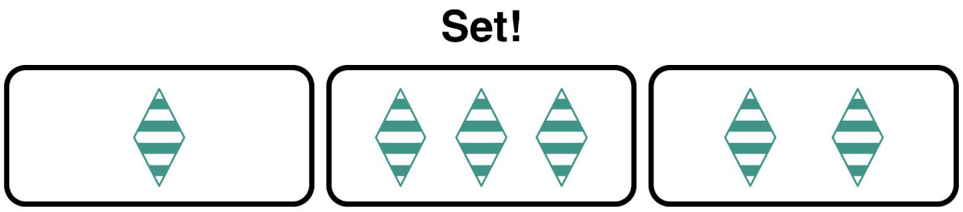
4. `ul > li`

_____

5. `li > *`

_____

1.B. Take a Set Before You Go. (CSS Writing, 10pts)



The above screenshot can be created by adding CSS to the following HTML:

```
<body>
  <h1>Set!</h1>
  <div class="row">
    <div class="card">
      <img src="green-diamond-striped.png" alt="green striped diamond" />
    </div>
    <div class="card">
      <img src="green-diamond-striped.png" alt="green striped diamond" />
      <img src="green-diamond-striped.png" alt="green striped diamond" />
      <img src="green-diamond-striped.png" alt="green striped diamond" />
    </div>
    <div class="card">
      <img src="green-diamond-striped.png" alt="green striped diamond" />
      <img src="green-diamond-striped.png" alt="green striped diamond" />
    </div>
  </div>
</body>
```

Write the CSS to add styles to this HTML to meet the expected output with the following appearance details. Remember to set flex properties to meet the screenshot output:

- Card positioning:
  - The horizontal spacing for the three cards is achieved by setting the appropriate flex property in the appropriate CSS selector to the `space-between` value.
- Card details:
  - The divs containing card images are each 200px wide and 75px tall with a solid black border of 0.35em width and a border radius of 15px.
  - The horizontal spacing for the images in each individual card is achieved by setting the appropriate flex property in the appropriate CSS selector to the `space-around` value.
  - All images should have a specified height of 60px and be vertically centered in each card.
- Heading:
  - The Set! heading is Helvetica font with a default of sans-serif and is centered on the page with 5px of spacing between the text and the top border of the cards

Complete your answer on the next page.

Write your answer to Problem 1.B. here.

2. Short Answers (18 points)

1. HTML/CSS. Consider the following HTML used to render a styled paragraph on a webpage:

```
<div class=”paragraph”>
  <font face="Arial">Welcome to our cookie store!</font>
  You will <b>never</b>, <i>ever</i>, <u>EVER</u> beat
  <font size="154px" color="purple">OUR</font> recipes!
</div>
```

a. What is one issue with this HTML based on what we've learned in Module 1 (HTML/CSS)? (1pt)

b. Describe 1 specific change you would make to improve the provided HTML. (1pt)

2. JavaScript/CSS. While we have discussed the benefits of using classes (e.g. `.hidden`) and CSS to style elements on a page, we have also shown a few cases where it is better to set the style of an element using `element.style.<property>` in JavaScript. Give an example of such a case. (1pt)

3. Asynchronous Requests. Consider ***a single function*** that makes ***multiple requests to different web services*** in JavaScript. Assuming each request does not depend on a response of another request, what is the benefit of these requests being made asynchronously (all at once) vs. synchronously (one after another)? (1pt)

4. Event Listeners. Consider the following JS program:

```
(function() {
  "use strict";
  window.addEventListener("load", initialize);

  function initialize() {
    $("my-btn").addEventListener("click", foo);
    $("my-btn").addEventListener("click", bar);
    $("my-other-btn").addEventListener("click", bar);
  }

  function foo() {
    console.log("Hello!");
    $("my-btn").removeEventListener("click", foo);
  }

  function bar() {
    console.log("Hi!");
  }
})();
```

Assuming the `#my-btn` and `#my-other-btn` buttons exist in the corresponding HTML, **circle** which of the four options would be correct console output for the following sequence of events: (1pt)

1. A user clicks the `#my-btn` button
2. A user clicks the `#my-other-btn` button
3. A user clicks the `#my-btn` button

a	b	c	d
Hello! Hi! Hi! Hello! Hi!	Hello! Hi! Hi! Hi!	Hello! Hi! Hello! Hi! Hi!	Hi! Hi! Hi! Hi!

5. Data Storage Trade-Offs.

a. It is possible to store/process data on a server using .txt or .json files. What is one advantage of using SQL databases to store data instead? (1.5pt)

b. Recall that `localStorage` can be used to store data on a user's browser. What is one example where it would be more appropriate to store data with `localStorage` instead of SQL?: (1.5pt)

6. GET vs. POST. In 2-3 sentences, identify a specific example where it is better to use a POST request instead of a GET request and justify your choice. (2pt)

7. PDO Connections. What is the purpose of a PDO object in a PHP web service? (1pt)

8. PHP/SQL Security. Consider the following 2 options for inserting a new todo item into a todos database:

Option 1:

```
$db->query("INSERT INTO Todos (todo, author, type)
            VALUES ('{$todo}', '{$author}', '{$todo_type}')");
```

Option 2:

```
$str = "INSERT INTO Todos (todo, author, type)
        VALUES (:todo, :author, :todotype)";
$params = array("todo" => $todo, "author" => $author, "todotype" => $todo_type);
$stmt = $db->prepare($str);
$stmt->execute($params);
```

Which option is more secure? Justify your choice. (1.5pt)

9. Regex. For each of the two regular expressions, circle all the string(s) below that match it (2pts):

i. `/^[A-Z]+4\d\d$/`

- `CSE431`
- `CSE154`
- `http400`
- `ABC4dd`
- `404`

ii. `/^pup.*y.jpg$/`

- `puppy.jpg`
- `^puppypuppy.jpg$`
- `pupy.jpg`
- `puppyparty.jpg`
- `puppykitty.JPG`

Regex reference:

[abc]	A single character of: a, b, or c	.	Any single character	(...)	Capture everything enclosed
[^abc]	Any single character except: a, b, or c	\s	Any whitespace character	(a|b)	a or b
[a-z]	Any single character in the range a-z	\S	Any non-whitespace character	a?	Zero or one of a
[a-zA-Z]	Any single character in the range a-z or A-Z	\d	Any digit	a*	Zero or more of a
^	Start of line	\D	Any non-digit	a+	One or more of a
$	End of line	\w	Any word character (letter, number, underscore)	a{3}	Exactly 3 of a
\A	Start of string	\W	Any non-word character	a{3,}	3 or more of a
\z	End of string	\b	Any word boundary	a{3,6}	Between 3 and 6 of a

options: i case insensitive m make dot match newlines x ignore whitespace in regex o perform #{...} substitutions only once

10. **PHP FIle I/O.** Suppose a directory has the following structure:

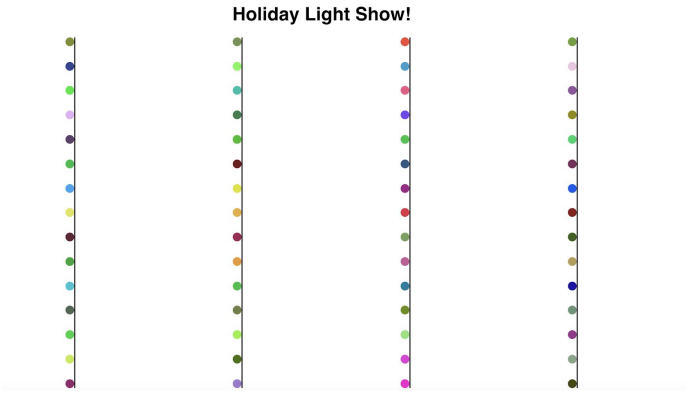
```
test.php
mydir/
    images/
        puppy.gif
        pupper.jpg
        doge.png
    lotsa-pups.txt
```

What do each of the following statements return if written in test.php? Use `[]` **notation for any arrays** and **put strings in ""**. (3.5pts)

Statement	**Return Value**
`scandir("mydir")`	
`glob("mydir/all-the-puppies.txt")`	
`glob("*/*pup*")`	

3. It's Time for the Holidays! (JS/DOM/Timers, 12 points)

It's the time of year when Seattle streets are filled with colorful lights! In this problem, you will create a holiday light show simulation using what you've learned in CSE 154 this quarter. There are two views on the Holiday Light Show page. Initially, the `#main-view` is shown and the `#lights-view` is hidden with the provided `hidden` class.



Initial (`#main-view`) Page Appearance	`#lights-view` during animation

Below is the provided HTML body your JS will be working with:

```
<body>
  <header><h1>Holiday Light Show!</h1></header>
  <section id="main-view">
    <p>
      How many light strands do you want to set up?
      <input type="number" id="count" /> (max of 10)
      <button id="start-show">Start Light Show!</button>
    </p>
    <p id="error-msg" class="hidden">Please enter a value between 1 and 10 (inclusive)</p>
  </section>
  <section id="lights-view" class="hidden"></section>
</body>
```

Setting up the Stage

When a user clicks the `#start-show` button, you'll write JavaScript to set up a light show of vertical strands of holiday lights based on the current input value of `#count`. If a value for the `#count` input box is missing, is less than 1, or greater than 10, the `#error-msg` paragraph should be displayed by removing the `hidden` class (see provided HTML).

Example of error message shown for invalid input

Otherwise (the input *is* between 1 and 10) each strand should be created as a `div` element with the class of `strand` having 15 children divs, each with the class of `light`. Your code should assign each light its own random background color using the provided `getRandomColor()` function (**no background colors are defined in lights.css**).

After adding these lights to each strand, the strand should be added as a child to the `#lights-view` section. So if a user has input "4" when clicking the `#start-show` button, 4 `.strand` divs will be created for a total of 60 `.light` divs. An example of this generated HTML hierarchy for `#lights-show` is as follows:

```
<section id="lights-view">
  <div class="strand">
    <div class="light"></div>
    ... 14 more lights
  </div>
  ... more strands depending on input box value
</section>
```

After creating and adding the strands to the `#lights-view`, the `hidden` class should be removed from this view and added to the `#main-view`.

Animating the Light Show

As soon as the `#lights-view` is displayed, the animation should start. Every 0.5 seconds, a new random background color should be given to each light. Below is a snapshot of an animated show, where 4 strands have been created:

You may assume adding `.strand` divs to the `#lights-view` and `.light` divs to each `kstrand` will result in the correct layout with provided CSS (you are not to write any CSS). You are to use the `hidden` class to hide/show any elements as specified.

You will write your JS in three parts (you may complete them in any order you prefer, and you may assume the two functions for b) and c) are correctly implemented if you call them in part a).

a) Write any page setup/initialization code.
b) Write a function to check input value and if valid, create the strands of lights and switch views.
c) Write a function to implement the light show animation.

For full credit, all of these parts must be implemented such that the program works as specified in the previous two pages. You may assume that the aliases `$(id)`, `qs(el)`, and `qsa(sel)` are defined for you and are included as appropriate.

```
(function() {
  "use strict";
  // provided constants
  const LIGHTS_PER_STRAND = 15;
  const INTERVAL = 500;

  window.addEventListener("load", initialize);

  // Provided getRandomColor function to use to return random hex colors for lights.
  function getRandomColor() {
    let result = "#";
    let hexValues = "0123456789ABCDEF";
    for (let i = 0; i < 6; i++) {
      result += hexValues[parseInt(Math.random() * hexValues.length)];
    }
    return result;
  }

  // Your solutions to Parts a, b, and c. will be written on the following pages.
```

```
/**
 * Part a. Write the page initialization code/functions. You may assume the functions
 * from Part b. (createLights) and Part c. (startAnimation) are implemented correctly.
 */
function initialize() {









}

/**
 * Part b. Write a function createLights to create strands of lights based
 * on the user input to the #count textbox. If there is not a value entered between 1
 * and 10 (inclusive) in #count, show the #error-msg paragraph. Otherwise, the
 * #lights-view should be shown and #main-view should be hidden at the end of this function.
 */
function createLights() {









}
```

```
  /**
   * Part c. Write a function startAnimation to start the light animation for
   * the created strands of lights. Assume the #main-view is hidden and #lights-view
   * is shown, and the strands with lights are already populated in #lights-view.
   */
  function startAnimation() {









  }

  // Any (optional) helper functions you write should go here.

})();
```

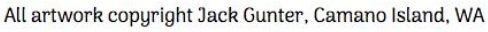
4. AJACKSketch (JS/AJAX, 20 points)

Jack Gunter is an eclectic artist who lives in Camano Island, WA. He has many talents: writer, filmmaker, and antique collector, but he is most known for his distinctive style of painting. Jack paints with brush strokes of egg tempera paints to form a rich and unique image on his canvases. We recently put together a quick website for Jack to highlight some of his work.

Implementation Requirements

When the page is initially loaded it will look like the image on the left - a drop down (`#works-list`), that that will be populated with types of Jack's works, and an empty section (`#works`). The HTML for this page is shown below.

The HTML page will link to `ajacksketch.js` which contains code you will write to add behavior to this page. The JS will make GET requests to `ajacksketch.php` (the same web service you will implement in Problem 5), to retrieve data that will be incorporated onto the page. For the purposes of this problem you may assume `ajacksketch.php` functions as expected.



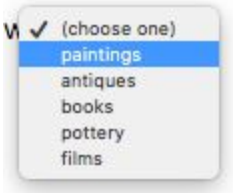
The body of the provided ajacksketch.html file is provided below:

```
<body>
   <header>
      <h1>Jack Gunter, Artist</h1>
   </header>
   <main>
      <section>
         Select types of work to view:
         <select id="works-list">
            <option value="">(choose one)</option>
         </select>
      </section>
      <section id="works">
         <h2 id="description"></h2>
         <div id="works-displayed">
         </div>
      </section>
   </main>
   <footer>
      <p>All artwork copyright Jack Gunter, Camano Island, WA</p>
   </footer>
</body>
```

This webpage will function as follows:

- When the page loads your JS will make the GET request `ajacksketch.php?mode=works` which will return the types of works as plain text. Currently this request returns the following:

  ```
  paintings
  antiques
  books
  pottery
  films
  ```

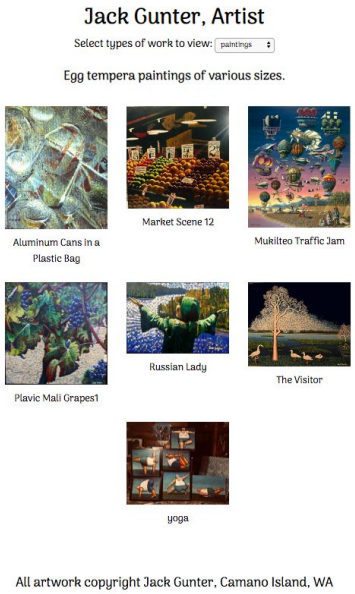
  

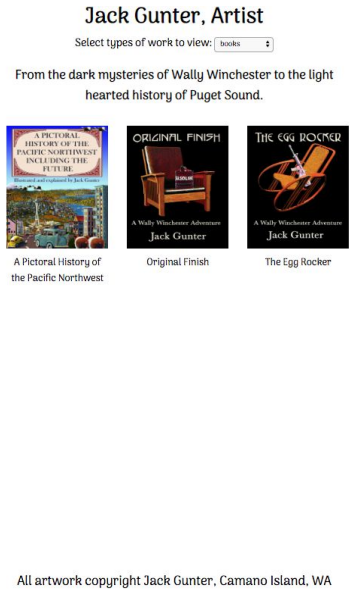
  These types of works are to be added as option elements in the drop down box `#works-list` (see figure to the right).

- Selecting a type of work from the `#works-list` should make a second GET request to `ajacksketch.php` with `mode=items` and a `type` parameter set to a value for the type of work to be retrieved. For instance, if the user wanted to retrieve all of Jack's paintings, the request would be made to `ajacksketch.php?mode=items&type=paintings`. The JSON formatted response will look like this example:

```
{
   "type": "paintings",
   "description": "Egg tempera paintings of various sizes.",
   "items": [
      {
         "name": "Aluminum Cans in a Plastic Bag",
         "image": "paintings/Aluminum_Cans_in_a_Plastic_Bag/picture.png"
      },
      {
         "name": "Market Scene 12",
         "image": "paintings/Market_Scene_12/image.jpg"
      },
   ...
   ]
}
```

Examples of the page displaying the paintings (left) and books (right) as type of works are shown below.





- The description of the type of work you are retrieving should be displayed in the `#description h2` element on the page.
- Each item should be displayed in a `div` created with an image (with alt text as the item name) and a paragraph (also containing the item name). These `div`s should be put in the `#works-displayed div`. Were you to inspect a page full of these items, each item on the page should have the following HTML after being created (where the blanks are filled in with the appropriate information from the JSON data):

```
<div>
   <img src="_______" alt="_______">
   <p>_______</p>
</div>
```

- If the user changes which type of work is selected in the drop down, the description and all of the items on the page should change to reflect the newly selected option.
- You should handle any fetch errors with `console.log`.

Write your JS solution below. Your code will be written in 3 parts:

a) Initialization
b) Fetching and displaying the drop down information
c) Fetching and displaying the contents of the page based on the user's selection

You may assume that the `checkStatus` function and the aliases `$(id)`, `qs(el)`, and `qsa(sel)` are defined for you and are included as appropriate.

```
(function() {
  "use strict";
  const URL = "ajacksketch.php";

  window.addEventListener("load", initialize);

  /**
    * Part a. Write your initialization code here. Note it may help to finish parts b
    * and c first.
    */
  function initialize() {









  }
```

```
/**
 * Part b. Write your function(s) to fetch and populate the drop down for
 * the web page here:
 */
```

```
// Part c should be implemented on the next page
```

```
  /**
   * Part c. Write your function(s) to fetch and display the contents of the
   * page based on the user's selection here:
   */
```

```
})();
```

5. A Portfolio in PHP (PHP Web Service, 20pts)

In this question, you will *implement* the PHP web service (`ajacksketch.php`) you were asked to use in the previous problem (JavaScript/AJAX). Note: there are additional features in this web service that were not used in the previous problems

Directory Structure and File Format Details

Your web service will be located in the same level as one file, `works.txt`, that has a list of the artists works, and a set of subdirectories that each correspond to the type of work the artist has done.

The subdirectories contain a file, `description.txt`, which contains a description of this type of work, and another level of subdirectories, one for each item Jack has has created or collected in that category. Each item subdirectory is named for the item, with underscores (`'_'` instead of spaces in the name), and contains a `picture.jpg` image file.

ajacksketch Directory Structure:

```
ajacksketch.php
works.txt
antiques/
   description.txt
   Stained_Glass_Light/
      picture.jpg
   Stickley_Chair/
      picture.jpg
  …
paintings/
   description.txt
   Aluminum_Cans_in_a_Plastic_Bag/
      picture.jpg
   Market_Scene_12/
      picture.jpg
  …
films/
   description.txt
   Siberian_Rescue/
      picture.jpg
```

`works.txt` File Contents:

```
antiques
paintings
films
books
pottery
```

`description.txt` File Contents (sample from `antiques/` directory):

```
An eclectic collection of antique furniture and
lighting.
```

Web Service Implementation

Your web service should support the following two GET requests (you may assume all query parameters passed are in lowercase):

`ajacksketch.php?mode=works`

This request should return a plain text response of all types of works that Jack currently has offered on the site, corresponding to the information in the works.txt file. Each work option should be output on a new line. The order of lines in the output does not matter.

A request to `ajacksketch.php?mode=works` should output the contents of `works.txt`, trimming any white space for each line. The output for the example contents would be:

```
antiques
paintings
films
books
pottery
```

`ajacksketch.php?mode=items&type=<type of work>`

This request should return a JSON response containing all of the information for each item under the type of work listed (identical to the example we gave in Problem 4):

```
{
   "type": "paintings",
   "description": "Egg tempera paintings of various sizes.",
   "items": [
      {
         "name": "Aluminum Cans in a Plastic Bag",
         "image": "paintings/Aluminum_Cans_in_a_Plastic_Bag/picture.png"
      },
      {
         "name": "Market Scene 12",
         "image": "paintings/Market_Scene_12/image.jpg"
      },
   ...
   ]
}
```

Error Handling

Your web service should handle the following errors with 400 error codes (in order of highest priority to lowest priority). If any error occurs, only output the respective error message in plain text.

- If missing the `mode` parameter or the `mode` parameter is passed with a value other than `works` or `items`, output an error with a message ”Please pass a mode of works or items”.
- If instead `mode=items` is passed you will need to check for the following:
    - If no `type` parameter is passed, output a message ”Missing type parameter for item request”
    - If a `type` parameter is passed but that type is not in our list of works, nothing should be output.

You are to write your solution to `ajacksketch.php` on the next pages in three parts:

a) Write a function to print an error message in the correct format with the correct header and end the program
b) Write a function to get all of the information for a particular type of work (ie, the description, and all of the items)
c) Write the main part of your `ajacksketch.php` service using part a) and b) (you may assume code from part a) and b) work)

Part a:

Write a function `error_message` that will handle the error messages for your system. This function should accept one string parameter, `$msg` and it will set the 400 Invalid Request error code and the output type to plain before printing `$msg` as the error message and stopping the script from running any further.

Part b:

Write a function `get_all_item_info` that takes a string parameter `$type` and returns an associative array based on `$type` where you may assume `$type` is listed in `works.txt`. The returned array should contain information found from the `description.txt` file and the information in each item subdirectory. For example, if `$type` has the value "films", the function might return the an associative array corresponding to the contents in the `films` directory:

```
{
  "type": "films",
  "description": "Mockumentaries that are actually pretty good.",
  "items": [
     {
        "name": "Siberian Rescue",
        "image": "films/Siberian_Rescue/picture.jpg"
     }
  ]
}
```

IMPORTANT HINT: You will want to use this help function `get_item_info($type, $item)` to get all of the information for one individual item in a category of work (for instance, just getting the information about the "Siberian Rescue Item")

```
/* Function to get all of the information about a particular item based on the $type of
 * work and a name of the $item's directory (with '_' characters between words). The
 * function will return an associative array with information about that item.
 * For instance to get the item info about the
 * films/
 *    Siberian_Rescue/
 *       picture.jpg
 *
 * The call to get_item_info("films", "Siberian_Rescue") will return
 *     array("name" => "Siberian Rescue", "image" => "films/Siberian_Rescue/picture.jpg")
 */
function get_item_info($type, $item) {
  $obj = array("name" => str_replace("_", " ", $item),
               "image" => "$type/$item/picture.jpg");
  return $obj;
}


/* Complete the code for the following function below for Part b. */
function get_all_item_info($type) {




}
```

Part c:

Assume your functions from Parts a and b work. In this part, you will write the rest of the `ajacksketch.php` web service, handling the request types specified on the previous pages. Hint: You will find it helpful to use the `get_all_item_info` function for the `mode=items` to get each of the the items for each type of work as part of the response.

```
<?php
# Write your code here:
```

```
function get_all_item_info($type) {
  # assume this function works as specified in Part B
}

function error_message($msg) {
  # assume this function works as specified in Part A
}
?>
```

6. Around The World (SQL, 15pts)

Recall the following tables in the world database:

world:

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...							

countries

Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, capital, code2

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...			

languages

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...				

cities

a. Write a SQL query to list the **name** and **gnp** of all countries in Asia with a gnp greater than 123456.

Expected results (9 rows total):

name	gnp
India	447114.00
Iran	195746.00
Japan	3787042.00
...	

Write your SQL query below:

b. Write a SQL query to list the **name of all countries** having a population of fewer than 85,000 and which have languages spoken (but not necessarily official) by a percentage of at least 50% of the population. Include the **language name** and **language percentage** in your results. Order the results by language name (alphabetically increasing) breaking ties by percentage (decreasing). *Hint: The percentage column in the languages table represents the percentage of the population speaking that language in the corresponding country.*

Expected results (16 rows total):

language	name	percentage
Creole English	Dominica	100.0
Creole English	Saint Kitts and Nevis	100.0
Creole English	Antigua and Barbuda	95.7
English	Bermuda	100.0
English	Gibraltar	88.9
Faroese	Faroe Islands	100.0
...	...	...

Write your SQL query below:

c. Write a SQL query to list the **city name** and **continent name** of all cities containing the word “sea” which are cities in countries having English as an official language. Order the results by city name (increasing alphabetically).

Expected results (3 rows total):

name	continent
Seattle	North America
Southend-on-Sea	Europe
Swansea	Europe

Write your SQL query below:

7. Extra Credit

For this question, you can get 1 point of extra credit (demonstrating at least 1 minute's worth of work and being appropriate in content). You can draw or write your answers in text.

If you could give your TA a surprise winter break gift, what would it be?